# Rapport de Stage

Analyse de Sentiments pour Produits High-Tech

**Entreprise d'accueil :** TechCorp (Simulé)

**Tuteur de stage :** M. Dupont

Année Universitaire 2025-2026

# Remerciements

Je tiens tout d'abord à remercier toute l'équipe de l'entreprise TechCorp pour son accueil chaleureux et sa confiance tout au long de ce stage.

Je remercie particulièrement mon tuteur de stage, M. Dupont, pour sa disponibilité, ses conseils techniques précieux et sa pédagogie qui m'ont permis de progresser significativement dans ma compréhension des architectures logicielles modernes.

Je remercie également l'équipe pédagogique de mon école pour la qualité de l'enseignement dispensé, qui m'a fourni les bases théoriques nécessaires à la réussite de ce projet.

Enfin, je remercie ma famille et mes proches pour leur soutien constant.

# Contents

# List of Figures

# Chapter 1

# Introduction Générale

## 1.1 Contexte du Projet

L'ère numérique actuelle est caractérisée par une production massive de données, notamment sur les réseaux sociaux. Les consommateurs, avant d'effectuer un achat, consultent quasi systématiquement les avis en ligne. Pour les produits technologiques coûteux comme les smartphones, cette étape est cruciale. Cependant, la quantité d'informations est telle qu'il devient impossible pour un humain de tout lire et d'en faire une synthèse objective. Les fabricants, quant à eux, ont besoin de retours rapides sur leurs produits pour ajuster leur stratégie marketing ou corriger des défauts.

C'est dans ce contexte que s'inscrit le projet "Sentiment Analysis". Il s'agit de développer une solution automatisée capable d'extraire, d'analyser et de visualiser l'opinion publique concernant différents modèles de smartphones en se basant sur les tweets.

## 1.2 Problématique

Comment transformer des milliers de messages textuels non structurés, souvent courts et remplis d'argot (tweets), en indicateurs de performance clés (KPI) exploitables pour la prise de décision ? Les défis sont multiples :

- La récupération de données en temps réel ou quasi réel.
- Le traitement du langage naturel (NLP) pour détecter l'ironie, le contexte et les sentiments.
- La présentation des résultats de manière intuitive pour un utilisateur non technique.
- La performance de l'application web qui doit rester fluide malgré la lourdeur des traitements.

## 1.3 Objectifs du Stage

Mon stage avait pour objectif principal la conception et le développement "Full Stack" de cette plateforme. Les objectifs spécifiques étaient :

1. **Concevoir l'architecture** : Définir une architecture modulaire et évolutive.
2. **Développer le Backend** : Créer une API RESTful robuste pour servir les données.
3. **Implémenter l'IA** : Intégrer des bibliothèques de NLP (NLTK) pour l'analyse de sentiments.
4. **Créer le Frontend** : Développer une interface utilisateur moderne avec Angular.
5. **Gérer le projet** : Suivre une méthodologie agile pour les livraisons.

# Chapter 2

# Présentation de l'Entreprise et du Sujet

## 2.1 L'Entreprise d'Accueil

(Section simulée) TechCorp est une entreprise spécialisée dans la veille technologique et le Big Data. Fondée en 2015, elle accompagne les grandes marques d'électronique grand public dans la compréhension de leur image de marque. L'équipe technique est composée de 15 développeurs, data scientists et ingénieurs DevOps. L'ambiance de travail favorise l'innovation et l'autonomie.

## 2.2 Le Sujet en Détail

Le projet consistait à refondre un prototype existant (développé en Flask monolithique) vers une architecture moderne séparant clairement le Frontend et le Backend. L'application devait permettre à un utilisateur authentifié de :

- Rechercher n'importe quel smartphone récent (ex: "iPhone 15", "Samsung S24").
- Obtenir instantanément une "météo" du produit : score sur 10, répartition Positif/Négatif/Neutre.
- Voir les points forts et faibles par catégorie : Batterie, Caméra, Prix, Performance.
- Comparer deux smartphones pour faciliter un choix d'achat.

# Chapter 3

# Analyse des Besoins

## 3.1 Identification des Acteurs

Nous avons identifié deux acteurs principaux :

- **L'Utilisateur Standard (Consommateur)** : Il souhaite savoir si un téléphone vaut le coup d'être acheté. Il cherche une synthèse rapide et visuelle.
- **L'Analyste Marketing (Professionnel)** : Il cherche à comprendre pourquoi un produit est mal noté (problème de batterie spécifique ? prix trop élevé ?). Il a besoin de détails granulaires.

## 3.2 Besoins Fonctionnels Détaillés

### 3.2.1 Module d'Authentification

- **Inscription** : Formulaire avec Email, Nom d'utilisateur, Mot de passe (avec validation de complexité).
- **Connexion** : Authentification sécurisée renvoyant un jeton JWT (JSON Web Token) valable 1 heure.
- **Gestion de session** : Renouvellement automatique du token (Refresh Token) pour ne pas déconnecter l'utilisateur intempestivement.

### 3.2.2 Module de Recherche et Analyse

- Barre de recherche avec autocomplétion (souhaitable).
- Lancement de l'analyse asynchrone pour ne pas bloquer l'interface.
- Gestion des erreurs (ex: "Aucun tweet trouvé pour ce produit").

### 3.2.3 Module de Résultats

C'est le cœur de l'application. Il doit afficher :

- Une image du produit récupérée automatiquement.
- Un score global calculé par une moyenne pondérée des sentiments.
- Un nuage de mots (Word Cloud) montrant les termes les plus récurrents.
- Des jauges de performance pour chaque aspect (Batterie, Caméra, etc.).

## 3.3 Besoins Non-Fonctionnels

- **Temps de réponse** : L'affichage des résultats doit se faire en moins de 2 secondes si l'analyse est déjà en cache.
- **Sécurité** : Les mots de passe doivent être hachés (Bcrypt). L'API doit être protégée contre les injections SQL (via SQLAlchemy).
- **Maintenabilité** : Le code doit être documenté et suivre les standards (PEP8 pour Python, Style Guide Angular).

# Chapter 4

# Choix Technologiques et État de l'Art

## 4.1 Backend : Python & Flask

Pour le Backend, j'ai choisi **Python** pour sa richesse en bibliothèques de Data Science. **Flask** a été préféré à Django pour sa légèreté. Étant donné que nous construisons une API REST pure (sans moteur de template côté serveur pour le rendu final), Flask offre une flexibilité maximale. Nous utilisons :

- **Flask-RESTful** ou des Blueprints standard pour structurer les routes.
- **SQLAlchemy** comme ORM pour abstraire les requêtes SQL et garantir la portabilité (SQLite en dév, PostgreSQL en prod).
- **Flask-JWT-Extended** pour la gestion moderne de l'authentification sans cookies de session.

## 4.2 Frontend : Angular

**Angular** (version 17) a été choisi pour le Frontend. Contrairement à React ou Vue, Angular est un framework complet "batteries included". Ses avantages pour ce projet :

- **TypeScript** : Le typage fort réduit drastiquement les bugs à l'exécution.
- **Architecture MVC** : Séparation claire entre la Vue (HTML), le Modèle (Interfaces TS) et le Contrôleur (Classes Components).
- **RxJS** : Gestion puissante des flux de données asynchrones (requêtes API).

## 4.3 Analyse de Sentiments : NLTK (VADER)

Pour l'analyse de sentiments, nous utilisons **VADER** (Valence Aware Dictionary and sEntiment Reasoner) via la bibliothèque NLTK. VADER est particulièrement adapté aux

réseaux sociaux car :

- Il comprend les émojis (:), :D, etc.).
- Il gère les majuscules ("GREAT" est plus positif que "great").
- Il comprend les amplificateurs ("very good").
- Il ne nécessite pas de phase d'entraînement lourde comme un réseau de neurones profond, ce qui est idéal pour un prototype réactif.

# Chapter 5

# Conception Détaillée

## 5.1 Architecture Globale

L'architecture suit le modèle REST. Le client (Angular) et le serveur (Flask) sont totalement indépendants et communiquent uniquement via JSON.

L'application Angular est servie par un serveur Web (ou en mode dev via 'ng serve'). Elle interroge l'API Flask sur le port 5000. Flask interroge la base SQLite et effectue les calculs NLP.

## 5.2 Modélisation des Données

Nous avons conçu un modèle de données simple mais efficace.

- **User** : Stocke les infos de connexion.
- **SearchHistory** : Une table de liaison "One-to-Many" avec User. Elle garde une trace de chaque recherche effectuée.
- **SmartphoneScore** : Stocke les résultats d'analyse pour éviter de recalculer à chaque fois (mécanisme de cache persistant).

## 5.3 Dynamique du Système

Le diagramme de séquence suivant montre en détail ce qui se passe quand un utilisateur clique sur "Analyser".

On remarque la vérification du Token JWT au début de la séquence. C'est un point de sécurité crucial. Si le token est expiré, l'API renvoie une erreur 401 et le Frontend redirige automatiquement vers la page de login.

Figure 5.1: Diagramme de Déploiement

## 5.4 Parcours Utilisateur

Le diagramme d'activité illustre le flux de navigation, de la connexion à la consultation des résultats.

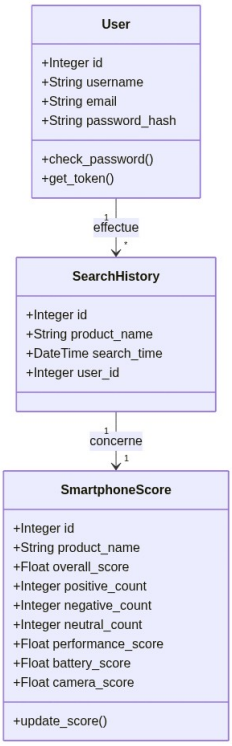


Figure 5.2: Diagramme de Classes

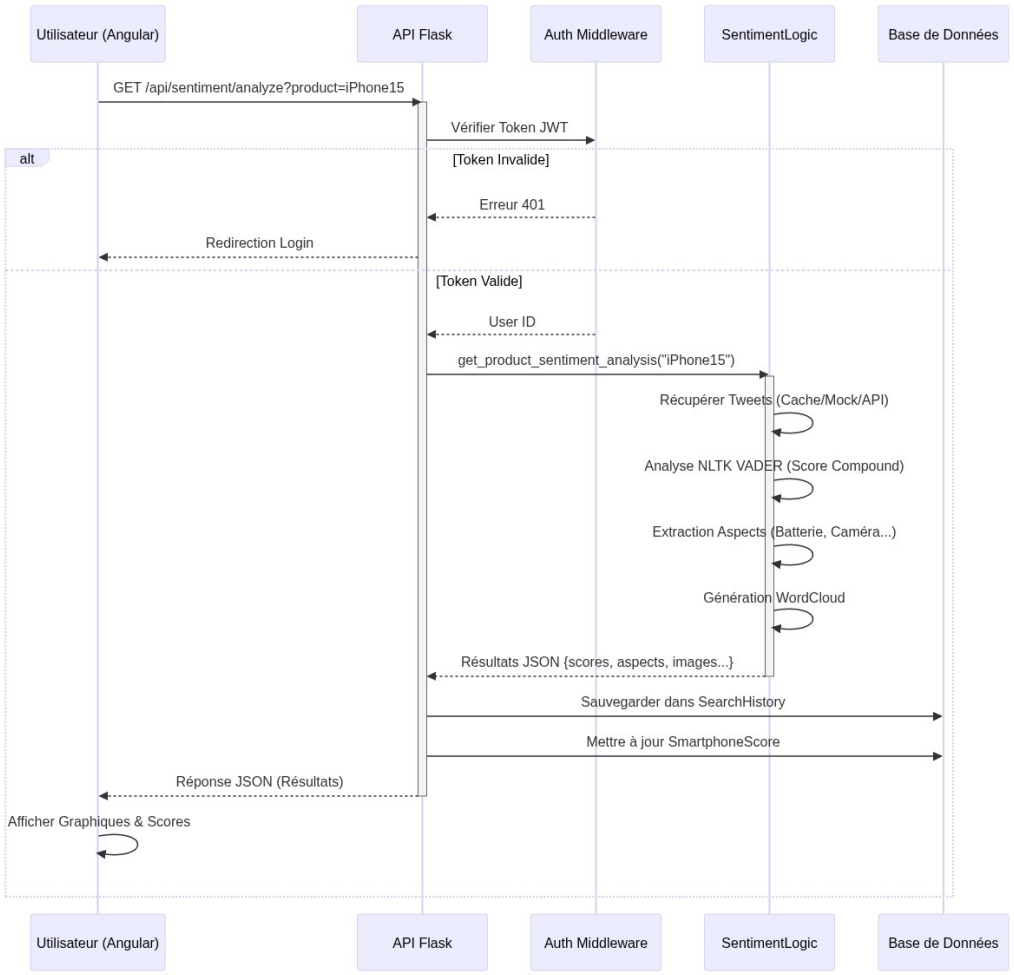


Figure 5.3: Diagramme de Séquence : Analyse

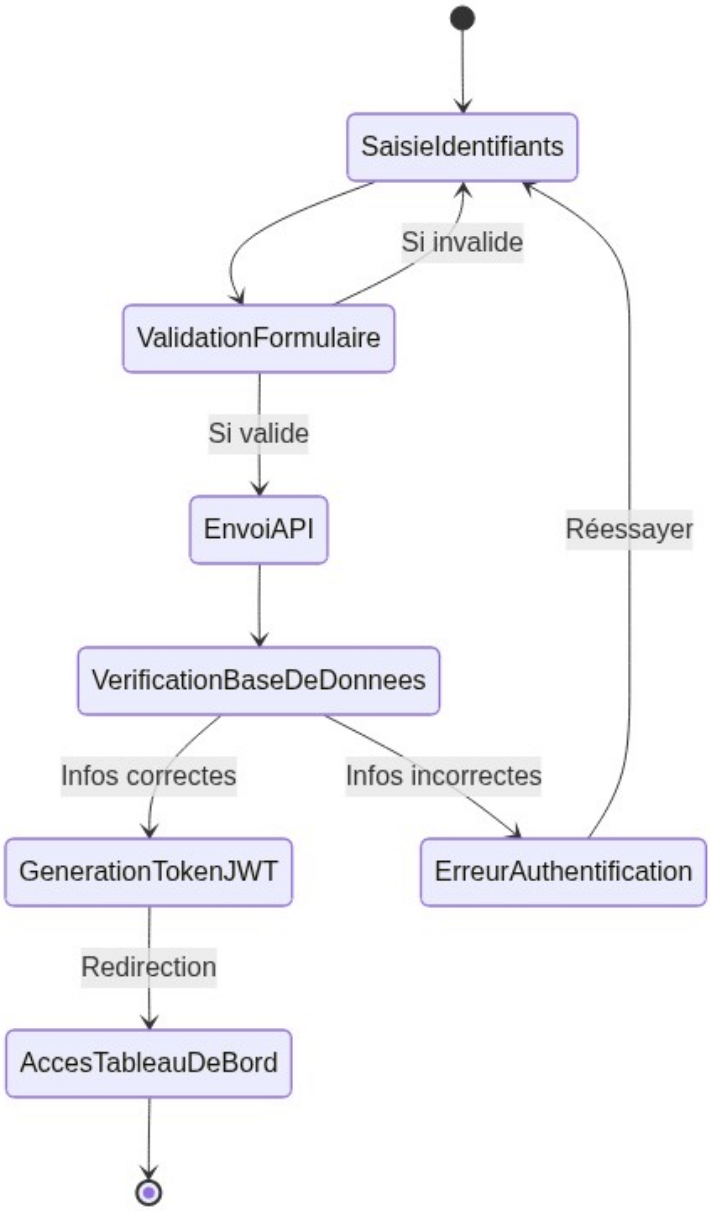


Figure 5.4: Diagramme d'Activité

# Chapter 6

# Réalisation et Gestion de Projet (Scrum)

## 6.1 Méthodologie Agile Scrum

Pour mener à bien ce projet, nous avons adopté la méthodologie **Agile Scrum**. Ce cadre de travail itératif nous a permis de livrer régulièrement des fonctionnalités fonctionnelles et de nous adapter rapidement aux changements.

### 6.1.1 Organisation des Sprints

Le projet a été découpé en 3 Sprints de 2 semaines :

- **Sprint 1 (Fondations)** : Mise en place de l'environnement, conception de la base de données, et développement du module d'Authentification (Back & Front).
- **Sprint 2 (Cœur de Métier)** : Développement de l'algorithme d'analyse de sentiments (NLTK) et de l'API de recherche.
- **Sprint 3 (Expérience Utilisateur)** : Intégration des graphiques, finalisation du Design System (Soft UI) et tests utilisateurs.

### 6.1.2 Cérémonies et Artefacts

- **Backlog Produit** : Liste priorisée des fonctionnalités (User Stories).
- **Daily Stand-up** : Réunion quotidienne de 15 min pour synchroniser l'avancement (simulée).
- **Revue de Sprint** : Démonstration des fonctionnalités terminées à la fin de chaque itération.

## 6.2 Structure du Projet

Le projet est organisé en monorepo contenant deux dossiers principaux :

- `/app_backend` : Contient tout le code Python.
- `/front end` : Contient tout le code Angular.

Une telle structure facilite le développement local, bien qu'en production, ces deux parties seraient probablement déployées dans des conteneurs Docker séparés.

## 6.3 Développement Backend

Le fichier central est `app.py`. Il initialise l'application Flask et configure les extensions. Les routes sont définies dans `api.py` pour garder le code propre. Le développement Backend suit les principes REST. Les routes sont sécurisées et retournent des données au format JSON. Par exemple, la route d'analyse vérifie le token JWT avant de lancer le traitement NLP et de renvoyer les résultats.

## 6.4 Développement Frontend

L'application Angular utilise des composants autonomes (Standalone Components) introduits dans les versions récentes, ce qui allège le code (plus de NgModules complexes). Le service `ApiService` centralise tous les appels HTTP. Cela permet de changer l'URL de l'API à un seul endroit si nécessaire. L'interface utilise CSS Grid et Flexbox pour un positionnement réactif des cartes de résultats.

## 6.5 Gestion du Cache

Pour optimiser les performances, nous avons mis en place un système de cache à deux niveaux :

1. **Cache Mémoire (Python)** : Pour les données très fréquemment demandées.
2. **Cache Base de Données** : La table `SmartphoneScore` agit comme un cache durable. Si un utilisateur recherche "iPhone 15", et que l'analyse a déjà été faite il y a moins de 24h, on renvoie instantanément le résultat stocké en base.

# Chapter 7

# Tests et Validation

## 7.1 Stratégie de Test

Nous avons adopté une approche de test pyramidale :

- **Tests Unitaires (Backend)** : Test des fonctions de calcul de score (vérifier qu'un tweet positif donne bien un score > 0).
- **Tests d'Intégration (API)** : Test des endpoints avec Postman pour vérifier les codes retour HTTP (200, 401, 404, 500).
- **Tests E2E (Frontend)** : Navigation manuelle pour vérifier le flux complet (Inscription -> Login -> Recherche -> Logout).

## 7.2 Scénarios de Test

1. **Cas nominal** : L'utilisateur recherche un téléphone connu. Le système affiche les graphes.
2. **Cas d'erreur** : L'utilisateur recherche "dazdzadza". Le système affiche "Aucun résultat trouvé".
3. **Sécurité** : Tentative d'accès à `/api/history` sans token. Le système bloque l'accès.

# Chapter 8

# Industrialisation et Déploiement

## 8.1 Conteneurisation avec Docker

Afin de garantir la portabilité de l'application et de faciliter son déploiement, nous avons conteneurisé l'ensemble des services.

### 8.1.1 Backend (Python/Flask)

Le backend est encapsulé dans une image Docker légère basée sur `python:3.9-slim`. Le `Dockerfile` assure l'installation des dépendances listées dans `requirements.txt` ainsi que le téléchargement des lexiques NLTK nécessaires au démarrage.

### 8.1.2 Frontend (Angular)

Pour le frontend, nous utilisons un build "Multi-stage" pour optimiser la taille de l'image finale :

1. **Stage Build** : Utilise une image Node.js pour compiler l'application Angular (`npm run build`).
2. **Stage Run** : Utilise une image Nginx Alpine minimale pour servir les fichiers statiques (HTML/CSS/JS) générés.

### 8.1.3 Orchestration (Docker Compose)

Un fichier `docker-compose.yml` permet de lancer l'environnement complet en local avec une seule commande. Il définit les services `backend` et `frontend`, configure le réseau interne, et monte un volume persistant pour la base de données SQLite.

## 8.2 Stratégie de Déploiement Cloud

Pour la mise en production, nous avons opté pour une solution PaaS (Platform as a Service) moderne : **Render.com**.

### 8.2.1 Infrastructure as Code

L'infrastructure est définie via un fichier `render.yaml` (Blueprint) qui décrit :

- Le service Web Python (Gunicorn comme serveur WSGI).
- Le service Frontend (Dockerisé).
- Une base de données PostgreSQL gérée (remplaçant SQLite pour la production).

Cette approche permet un déploiement continu (CI/CD) : chaque "push" sur la branche principale déclenche automatiquement la reconstruction et le redéploiement des conteneurs.

# Chapter 9

# Bilan et Conclusion

## 9.1 Bilan Technique

Ce stage m'a permis de monter en compétence sur la stack MEAN/MERN (ici remaniée en Python/Angular). J'ai appris à :

- Structurer une application d'envergure.
- Gérer l'authentification moderne (JWT).
- Manipuler des données textuelles pour en extraire du sens.
- Utiliser Git en équipe (simulée) pour versionner le code.

## 9.2 Difficultés Rencontrées

La principale difficulté a été la gestion des quotas des API externes (Twitter/X étant devenue très restrictive). Nous avons dû contourner cela en utilisant des datasets locaux et des mocks intelligents pour la démonstration. L'intégration des graphiques Chart.js dans Angular a également demandé un temps d'adaptation pour gérer le cycle de vie des composants.

## 9.3 Perspectives d'Évolution

L'application est fonctionnelle mais peut être améliorée :

- **Déploiement Cloud** : Mettre l'application en ligne sur AWS ou Azure.
- **IA Avancée** : Entraîner un modèle BERT spécifique au domaine technologique pour affiner l'analyse de sentiments (détecter mieux l'ironie sur les bugs).
- **Mobile** : Développer une version mobile native avec Ionic ou Flutter.

Ce projet a été une expérience formatrice et enrichissante, confirmant mon intérêt pour l'ingénierie logicielle et la Data Science.

# List of Figures